A Proactive Inferencing Agent for Desk Support

Hans-Peter Schnurr & Steffen Staab
University of Karlsruhe, Institute AIFB, 76128 Karlsruhe, Germany
http://www.aifb.uni-karlsruhe.de/WBS/
{schnurr,staab}@aifb.uni-karlsruhe.de
Tel.: +49-721-608 7363 Fax: +49-721-693717

Abstract

We describe an approach towards integrating the semantics of semi-structured documents with task-support for (weakly-structured) business processes and proactive inferencing capabilities of a desk support agent. The proactive assistance of the intelligent agent is motivated by the requirements posed in (weakly-structured) business processes performed by a typical knowledge worker. First, we introduce a reactive agent that provides knowledge out of an organizational memory for the business task at hand. The building of the reactive agent requires rather rigid query structures that do not fit nicely with varying precision of knowledge found in the organizational memory. Thus, we propose an enhanced agent that reasons proactively about what might be interesting to you and what might be due in your next step.

Introduction

Intelligent information agents are often compared against their human counterparts, such as secretaries or other colleagues. The comparisons usually find that human assistants

- observe what you do,
- think about what you do,
- have expectations about what you do next,
- thus cope with incomplete information, and still if you ask them they
- respond quickly to your questions.

Though all of these properties are highly desirable for your personal information agent, in order that they answer you fastly and that you save tedious working schemes, nowaday's agents lack most of these properties. We here present an architecture and techniques that (partially) implement these properties within a realistic setting, providing information agents that

- reason proactively about what you might be doing,
- take into account what information you have provided so far,
- build up reasonable models about what information you might deserve next, thus
- narrow down choices for potential answers, even if information is missing, and, hence,
- respond to you much faster and much more precisely.

Subsequently, we will first describe our motivation for building proactive information agents and describe the *Scenario* into which they are integrated. Then, we describe our starting point, viz. a *Reactive Agent Support*, that includes knowledge from an organizational memory created through the usual work tasks of the knowledge worker. On this basis we build our enhanced agent that improves the rather rigid scheme of requiring information from the user and querying the inference engine. Our *Proactive Inferencing Agent* builds on just the same basic modules, however it employs a refined inferencing strategy that allows for earlier, faster and still more frequent support than the reactive approach. Moreover, it reduces the burden on the human who models the information agent for a particular application and, thereby, balances automatically between the need of specific answers and the requirement to produce an helpful answer for a broad variety of possible input data. Some examples will elucidate what we pursue with and how we achieve this modeling balance.

Scenario

Our approach is embedded in a typical knowledge management setting. In such work desk settings, proactive information agents have often been devised as useful helpers that may facilitate the task at hand the knowledge worker tries to complete on her computer. For instance, project management involves the compilation of a project plan, allocating resources and people in an appropriate way. Naturally, the project manager has to compile a team from within a large company where she is hardly able to know everyone with his capabilities and experiences. Thereby, she must meet the following planning requirements:

- Participants must be available for the project,
- participants should have particular technical knowledge that is needed for the project, and

- they should have some knowledge about the client in this project or at least about a client with a similar industry background.

Currently, there are several approaches to handle this problem. First, the team might be compiled from a set of people the manager knows by chance. Second, project listings might circle in the company. Third, all the information might be maintained by a human resource department, e.g. in a central database. However, all these possibilities come with personnel overhead, time lags or a lack of quality.

Hence, for our scenario we have the following — realistic — assumptions. First assumption, the project manager compiles a project plan that she uses to estimate the man power and expertise she needs for the project, e.g. with a project planning software that supports the creation of network plans, common spreadsheet, or text processing software. What is important at this point is that persons who execute this task on a regular basis usually hold on to a particular tool and a particular way of executing this task with this tool. For instance, it is quite common that a project manager creates a template (or uses a template that is provided by her company) in order to execute and document the planning task. Second assumption, the information that she relies on is drawn from her personal knowledge, from the knowledge of people she asks, and from the knowledge available in other project documents and in the intranet. Naturally, only the third type of knowledge available digitally is the one that can always be accessed electronically and, thus, it is the one we want to exploit for our knowledge support mechanisms. A common document of this kind would be a project page describing the name, the goal, the participants and techniques used in the project.

Let us now assume that a groupware platform exists that handles scheduling tasks. The knowledge that is necessary in order to fulfill the three requirements mentioned at the beginning of this section can be found as follows: Availabilities may be retrieved from the scheduling database, and technical knowledge may be inferred from employees' participations in projects at project web pages. Obviously, it is very tedious, sometimes nearly impossible for the project manager to gather the information she needs from these different sources. In the next section, we will show how the knowledge for the project planning task may be provided automatically via Reactive Agent Support, once the project planning task has been analyzed and an appropriate methodology and IT support has been introduced to the enterprise.

The objective for our Proactive Inferencing Agent, sketched thereafter, is more akin to having a competent person around you who tutors your project planning. For this purpose, the agent is supposed to reason about what you might ask him next, while you keep on typing (or speaking) into your computer. For instance, you start your project plan with an abstract, noting title, client and time interval. Then you might want to search for people who might participate in the project. Thus, you might pose a question like "Who has experience with XML?" to the agent and you expect an answer almost before you actually pose your question — because Joe Doe is the only one who has experience with XML at all and you noted at the very beginning that you require this type of knowledge — maybe regardless of schedule conflicts.

Reactive Agent Support

In order to provide a concise picture of our approach, we give an overview of the main modules (cf. Figure 1) and capabilities, before we integrate these different building blocks into our Reactive Agent Support Framework. A detailed description of the main modules is given in (Staab & Schnurr, 1999).

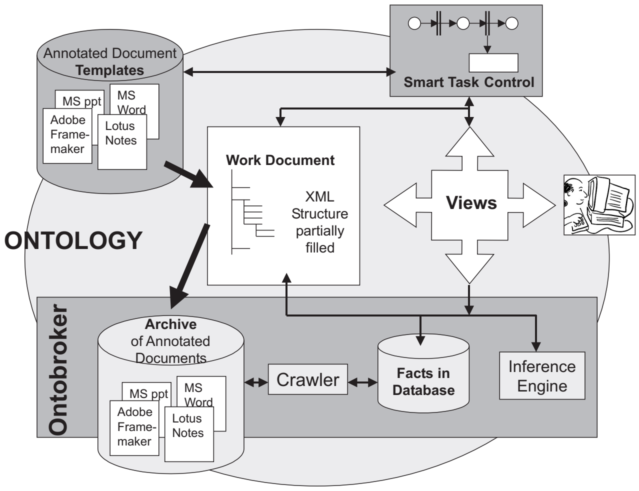

Figure 1: Reactive Agent-Support Framework

Ontobroker as Intranet Environment

Typical intranet environments comprise at least two technical services. First, they offer means to store documents. In particular, current technology tends to make the boundary between file servers and intra-/internet servers, on the one hand, and web pages and files, on the other hand, disappear. Hence, we may assume that all documents are available in the intranet. Second, intranet environments offer technology for navigating the intranet environment and finding information.

Our KM methodology builds on the framework given through the Ontobroker approach as described in (Benjamins, Fensel, & Pérez, 1998) and (Decker *et al.*, 1999). The main components of Ontobroker from a functional view point are, *(i)*, the underlying ontology, that defines concepts, attributes, relations and rules about a domain; *(ii)*, the annotated document sources, providing facts structured by annotations related to the underlying ontology; *(iii)*, a crawler that gathers facts from the documents and stores them into a database, and, *(iv)*, the inference and query engine that reasons on the database to derive answers.

Annotated Document Templates

Current business documents come in many different guises, such as letters, faxes, order forms, notifications, receipts, memos, private home pages, or project home pages. Nevertheless, it is quite common that these different forms are not chosen arbitrarily, but rather these forms are often standardized up to a certain point, indeed they often come with a particular semantics, such as the short notes that allow the reader to determine sender, reader, urgency, and further actions that need to be taken with a very short glimpse (e.g. check boxes for "please answer"). Similarly, letters are usually not allowed to come in a completely free form, but they are usually composed with a particular corporate identity in mind. This corporate identity defines fonts, but it also pervades the way a company presents itself to the outside world.

With our approach we go even one step further, since we also link these documents with corporate identity styles to the enterprise's ontology (or ontologies). This leads us to annotated document templates, and, thus, makes the contents of common business documents available for an explicit knowledge repository.

SGML (Standard Generalized Markup Language) and a subset of it, XML (eXtensible Markup Language), are standardization efforts that aim at a general scheme for exchanging documents. Given the widespread support among major computer software providers that XML has found recently, it is reasonable to assume that the structure of any business document will be accessible by way of XML annotation and query tools in the very near future.

In our scenario, this is also of particular interest, because SGML/XML gives us the power to reason about document structures and contents. For instance, the XML-tags in the pseudo document from Table 1, i.e. a document without actual facts, might serve as a template for project descriptions in general. The (XML) annotations describe the semantics (and, possibly, some layout) of the document structures. When the user fills in parts of the document (cf. Table 2) in order to complete her business task, then she connects the information she provides with corresponding metainformation. Thereby, XML structures may either be derived directly from the ontology (cf. Erdmann & Studer (1999)), or they may be manually specified and mapped onto the ontology.

Business Processes

The controlled interaction between the contribution to document contents and the performance of business tasks is of particular concern to an information agent that aims at the delivery of relevant information at the right time. Hence, we build on a special version of Petri nets, viz. so-called SGML nets by Weitz (1998), that allow to formulate the progress of the business process in terms of the document contents. Weitz's approach mostly aims at mechanisms that cover the comparatively rigid parts of the workflow, therefore his approach is not adequate to model all the — typically unordered — actions of a knowledge worker. Nevertheless, the scheme is most suitable to distinguish between different high-level, well-ordered business tasks, and, thus, to capture their specific *business contexts* and the views that are relevant during their execution by the worker (also cf. Staab & Schnurr (1999)).

For example, the high-level goal of project planning may be separated into several, distinct, well-ordered tasks (cf. Figure 2). Each of the tasks requires particular knowledge — and, hence, specific help from the intelligent assistant. The SGML-net mechanism allows to capture prerequisites for changing from one task to the other and to define task-specific views onto the organizational memory. Thereby, the current work document(s) reflect the task(s) that must be performed next, e.g. a project team needs to be compiled. This work involves communicating with prospective project participants. The problem often lies in identifying appropriate participants and in establishing the communication link. As for the first, our methodology allows the establishment of blueprint questions like "Which person in the company knows about X and has capacity for projects as of Y?" As for the second, given that a per-

Table 1: The XML structure for a project plan.

<project>		
<author>	</author>	
<plandate>	</plandate>	
<participants>		
	<member>	</member>
</participants>		
<Ganttchart>	</Ganttchart>	
<tasks>		
	<task>	</task>
</tasks>		
</project>		

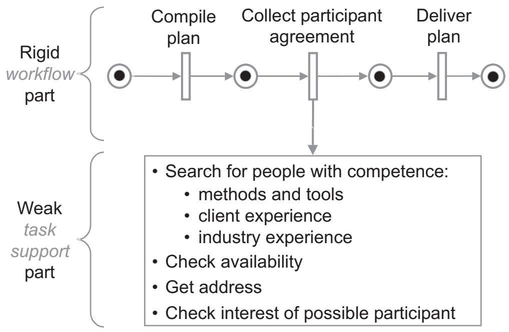

Figure 2: Integrating workflow and context-based views

Table 2: Filled template.

<project>			
<author>		Jill Dole	</author>
<plandate>		October 18th, 1999	</plandate>
<participants>			
	<member>	Jill Dole	</member>
	<member>	Hans-Peter Schnurr	</member>
	<member>	Steffen Staab	</member>
</participants>			
<Ganttchart>		here goes the table	</Ganttchart>
<tasks>			
	<task>	Analysis of Nordic Life Business Processes	</task>
	<task>	Analysis of Nordic Life Organigram	</task>
</tasks>			
</project>			

son is identified, the fax number(s) or e-mail adresses may be retrieved simply by giving the name information. Thereby, it is not required that the fax number is stated in a corresponding database entry. Rather it may only be given in the signature of a mail in the general accessible mail archives, or on a home page or only indirectly: via the group that this person belongs to and a rule that states an implication. Hence, via this context-specific questions the intelligent assistant supports the manager in planning her project.

To integrate the Context-based Views and SGML nets, we annotate transitions with logic predicates that define when a transition may be executed. In addition, we define logic predicates at transitions and at places that define context-based views. Thus, we allow to describe what major steps may follow subsequently and what views may be required to enable the completion of a particular step.

Example for Reactive Processing

Considering a concrete example aligned to our scenario, our tool has to support a project manager who compiles a team to implement some Knowledge Management Tool at an insurance company in May 2000. The first step for her is to find people in the company who know about the clients ERP Tool A, and have experience with XML, are available at that date, ideally have experience in the insurance industry and want to participate at that project. To ask the prospective team members about their interest in her project, she decides to send a fax. Therefore she starts the word processing software, opens the fax template "Contacting prospective team members and fills in the right column of the template form (cf. Figure 3).

The template serves as an input interface for the inference and query engine of the Ontobroker system. Several queries are modelled that depend on which template fields are filled (e.g., cf. left column in Figure 3). A click to the menu bar yields a list of possible queries and a click to to the corresponding query starts the inference engine. In our example, the project manager selects the query "get people with knowhow XML". The inference engine combines the available database information (gathered from annotated project web pages, project plans and home pages of employees; cf. Tables 3 and 4) and derives the answer: XML was used in the project "Xfiles". A rule in the ontology concludes that all the participants of a project have experience with methods of that projects. The person who fits that description ("Joe Doe") is placed in the header of the fax cover page.

The example shows also the drawbacks of the reactive agent support. Fortunately in the first example, the user finds exactly one answer. This would not be the case, if a user would ask for people with knowhow about ERP Tool A. This latter search would require more detailed modelling of a refined query. However applying a more specific query to the task at hand causes new problems in general, e.g., with the first example, since the corresponding, more specific, query for "XML and availability would not result in any answer at all. The careful balance between specificity and generality of queries is currently achieved by the user who selects from a list of queries. As one may easily see in Figure 3, this strategy puts too much burden on the user since she must choose from a long menu list of difficult and steadily changing queries. A more suitable and user friendly support would be to give hints about possible answers in a proactive manner. In addition, this reactive agent scheme requires time consuming modelling of a large number of queries. In our specific example, the engineer has to consider three interesting input fields (client, schedule, requirements) which lead to seven queries that may be of interest - more possibilities for input may even amplify the problem. In the following, we show an approach how to solve these two problems together.

Proactive Inferencing Agents

The application that we have just outlined provides additional assistance in your daily business processes. However, it is still very far from your favorite secretary some capabilities of whom we have listed in the introduction. In this chapter, we want to push the edge a

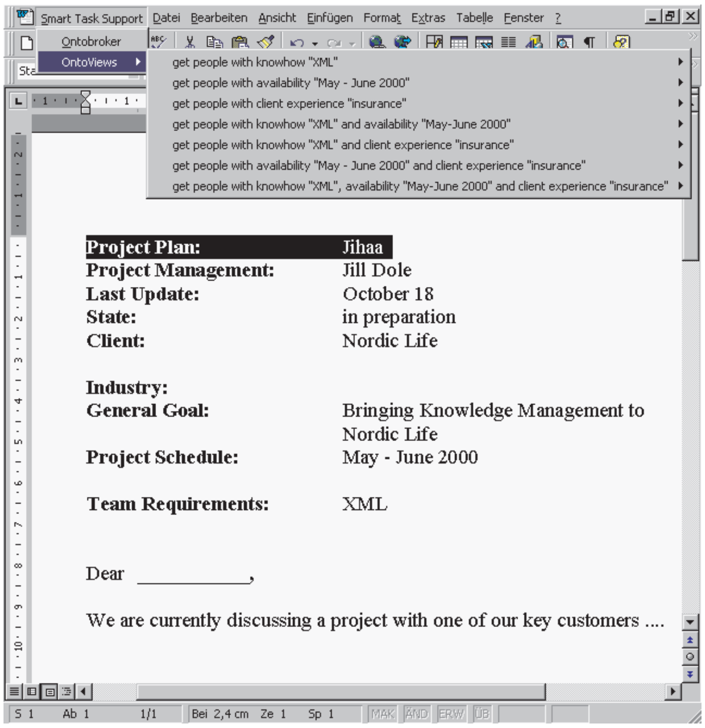

Figure 3: Fax template "Contacting prospective team members"

little further by adding a proactive component to our application. For this application, we first describe the inference engine that is employed in a little more detail, before we go on to extend its capabilities.

Inference Engine

The reasoning of the inference engine involves the following stages (cf. (Fensel, Angele, & Studer, 1998) for an elaboration):

1. The high-level modeling of ontology, facts and queries, which is all done in F-Logic, is translated into a set of horn clauses.[1]

2. The horn clauses are processed using a strategy called dynamic filtering (Fensel, Angele, & Studer, 1998). The implemented strategy may very well be conceptualized by a dynamic programming approach with an agenda as its central data structure (cf. Boddy (1991)). This means that inferencing subtasks are put onto the agenda, ordered according to a particular strategy, and then retrieved from this agenda for actual processing.

3. The inference engine — in contrast to simple Prolog interpreters — allows to store the facts that have once been derived in the database. Hence, if the same or similar queries are posed repeatedly, the computational load may be much lower than if the derivation must be started from scratch.

4. The variable bindings that were open in the query are returned by the inference engine, thus, yielding a set of tuples corresponding to all the facts in the semantic model that match the query.

Thus, the F-Logic inference engine by Angele and Decker combines ordering-independent reasoning in a high-level logical language with a well-founded semantics and a very flexible mechanism for modifying reasoning strategies that we will exploit in the following.

Proactive Inferencing

Let us now reconsider the example setting described in the reactive framework. From the inferencing point of view the key components were, *(i)*, an application that provides facts concerning the content and the context of a query, *(ii)*, precompiled queries, which will be executed if all the related facts are given, and, *(iii)*, a deductive database that may yield answers to queries.

Now, in order to proceed from a reactive to a proactive approach we must cope with, *(i)*, an application that defines the context only partially, and with, *(ii)*, semi-defined queries, where not all facts related to a query have been given yet, and, *(iii)*, we must provide a deductive database that takes advantage of precomputations in order to react quickly.

As we have found these are not requirements that contradict each other, but indeed with the proper inferencing strategy they complement each other quite nicely. The high level view onto the interaction of these needs is depicted in Figure 4: At the beginning we are given a set of facts and an ontology describing a set of concepts, relations, and rules. The facts that might be derived by applying all the rules in a forward-chaining manner would most often cause an overflow of data in the data storage — in fact the semantic model could even be an infinite one, as it may easily occur when functions are used in the representation language. On the other hand, we find that there is a small set of facts that is actually needed as background information for the knowledge worker. For instance, in one situation the knowledge sought from the database might be sketched in the form of a small rectangle (cf. "First answer sought"). Since neither the context nor the actual background knowledge is completely specified, the best one can hope from the inference engine is a set of facts that contain the knowledge required at this point of work, but that is not minimal with regard to the actual task the knowledge worker has to solve.

However, this need not even be a disadvantage. In fact, we have described above that formerly derived

[1] F-Logic is a frame-logic representation language conceived by Kifer, Lausen, & Wu (1995). In the implementation by Angele and Decker that we use, F-Logic is a proper subset of first-order predicate logic. Concepts and relations are reified and, hence, may be treated as first-order objects over which quantification is possible. For efficient processing, F-Logic is translated into a datalog-style representation (cf. Lloyd & Topor (1984), Decker (1998)).

Table 3: Knowledge about Projects

	Project	Participant	Purpose	Methods	Client
1.	"Xfiles"	Joe Doe	Knowledge Management Support with Ontobroker	XML, Ontobroker	Southern Insurance
2.	"Yawn"	Fred Bloggs, Walter Moe	Yet another accounting scheme	ERP Tool A	Kwik Fit
...	...	...	...	...	...
8.	"Guru"	Jane Smith	Great user utility through nothing	ERP Tool A	WonderTool

Table 4: Knowledge about Employees

Employee	Telefone	Fax	eMail	availability in 2000
Fred Bloggs	234567	991234	fred@onto.de	July - October
Joe Doe	345678	992345	joe@onto.de	August - December
Walter Moe	987654	993456	walter@onto.de	July - November
Jane Smith	456789	994667	jane@onto.de	April - November
...	...	...	...	

facts need not be recomputed by the inference engine, thus, the pre-computation of a set of facts may even be useful when one moves on to another setting and another, similar, task. Just think of a task where you must contact several people that are related to a particular topic, e.g. knowledge management. Their actual relationship to knowledge management may be computed from facts like "person A has written a paper about knowledge management", "person B has written a mail to me about knowledge management", etc. In a particular context you might wish to contact person B, because you are co-authoring a paper — and of course, it is much more likely that you co-author a paper with a person with whom you have contact than with a person whom you might not know at all. The inference engine will probably fail to employ this rationale, since its contextual knowledge is very limited. But when you try to contact person A later on anyway (say in order to organize a workshop) then the inference engine may take full advantage of previous computations — the second answer you seek is already contained in the first set of facts that have been computed before (cf. Figure 4).

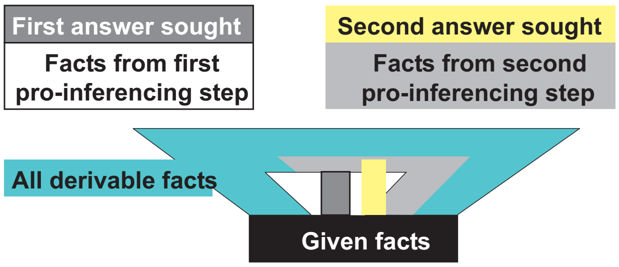

Figure 4: Model precomputation through pre-inferencing

Now the question remains open as to how the strategy just explained is realized in the actual application. The realization must take into account that model pre-computation is expensive and that the user may pose an explicit query at any point in time. Hence, one needs to avoid that model pre-computations overflow the data storage with useless facts and that the pre-computation actually prevents efficient query answering. To account for these stipulations we apply the following scheme:

1. The application queries the system when it determines that enough facts that belong to a query are known to the agent. In our setting this currently means that at least one open variable must be bound to a concrete fact, but in other settings more restrictive strategies might be necessary.
2. We distinguish between three different query priorities, viz. explicit queries by the user (priority 1), queries the variable of which are completely bound by the application (priority 2), and queries with unbound variables (termed "underspecified queries"; priority 3). Underspecified queries receive the lowest priority, they are preempted by the other two types of queries and they are terminated when their computation reaches a time limit. Explicit queries by the user receive the highest priority. They preempt priority 2 queries such that the computation of the latter may be continued after the completion of explicit query.
3. To allow for preemption and (if necessary) continuation of a query, the agenda strategy is modified. Entries on the agenda that are of priority 3 are cleared from the queue when a query of priority 1 or 2 arrives or when the deadline for a query expires. Entries on the agenda that are of priority 1 are put at the beginning of the agenda such that they are handled first, but such that the processing of entries of priority 2 may be resumed after their completion. By this way, the inference engine is adapted to yield an anytime algorithm (Boddy, 1991).
4. Queries with several unbound variables easily lead to an abundance of possible results, thus, deterio-

rating the specificity that one might gain from our approach compared to information retrieval-based schemes. Hence, it is important to restrict actual output to results that can be easily grasped by the user. This means we only provide hints proactively, if the result is definite, i.e. if in spite of the underdefined context and background knowledge only one possible answer may be found for a particular entry. Furthermore, we provide a selection of up to 7 possible choices, if we expect the user to continue with just the information that we computed in our approach.

All in all, this scheme allows the provision of power that we sketched above: it takes full advantage of pro-active inferencing through a multiple-priority approach. Pre-computation comes with an internal time limit, such that we gain a careful balance between the derivation of definite results from a weakly-defined context, the pre-computation of facts queried later, and the computational loads for processing and storage required from the system.

Example for Proactive Processing

As mentioned in Section 3.4, it is nearly impossible to model both, enough queries such that you get the answer that you are looking for and not too many queries such that the user and the modeler can both deal with the number of queries that are required for a particular task.

In contrast, an intelligent assistant adapts to the particular facts it finds in its knowledge base. It employs specific queries when it is appropriate to find out about who of your colleagues fits best into your work plan and it employs general queries when a too specific one would lead you into a dead end road — our proactive scheme is appropriate to realize just that. For instance, resuming our running example, a project manager may want to include someone with XML expertise. Assuming a content in the knowledge base like given through Table 3, the necessity to include someone with XML expertise requires that Joe Doe is contacted for a fruitful project. The problems in the reactive framework come from the fact that a specification like "an expert in XML who is available from Mai to June 2000" is so specific that in many, many cases an empty set of corresponding facts may be derived.

In the proactive framework the situation is somewhat different. Let us assume that the project manager wants to contact two persons. One who knows about XML and one who is an expert with ERP Tool A. Furthermore, let us assume that the project manager deals with a template that queries for people depending on their expertise and availability during a scheduled time frame. When the manager first plans for the position requiring XML knowledge, she mentions this in her planning template. In the reactive framework she would have to enter the time schedule, too - and find out that no such person exists or she would have to choose from a possibly long menu of queries. In the proactive framework, the inference engine starts with a low priority query. Since only one expert for XML may be retrieved at all, the corresponding instance — possibly with some explanation — may be returned even before the project manager may start to plan more details. Thus, the agent uses possible idle times of the user and the user gets an early feedback on her plans that is collected from the organizational memory, the axioms in the ontology and the proactive inferencing scheme.

When the project manager plans for someone with expertise in "ERP Tool A", there are two possibilities. If the number of experts is low, the agent returns the selection of experts in the field. Otherwise it does not react until the time schedule is specified such carefully that the choice is narrow enough to present it to the user. Hence, the careful balancing between overly specific and overly general questions is delegated to the agent and need not be hand-crafted. In the reactive framework, an overly specific question might result in no information at all — yielding no information about Joe Doe, while an overly general question might easily plague the user with an abundance of query lists that she does not want to cope with when she has only described a few specifications yet.

Related Work

Our work is an approach for proactive support for the knowledge worker with the help of intelligent agents. In the research field of information agents, numerous systems for very specific domains and tasks exist and also several publications classify and describe those systems (cf. Klusch (1998), Weiss (1999)). For example, Liebermann (1998) gives an overview of approaches to design intelligent information agents (Letizia, Remembrance Agent, LetsBrowse, Firefly, Butterfly, ExpertFinder, Tet-a-Tete, Footprints System) that provide active assistance in the process of finding and organizing information. These agents learn from interaction with the user and anticipate the users needs. This is supported with the analyses of statistical information of web pages and user profiles. In contrast, we embed our agent support in specific tasks of a workflow, defined by document stuctures that are related to ontological knowledge. So we mainly differ from those information agents in that we use semantic, and not only statistical information.

In the knowledge management area related to our approach, the distinction is not so clear cut and the surveys are not so numerous. Hence, we here give a more detailed roadmap of these: Our starting point has been a common intranet environment, in which all documents are put such that they are widely available for reuse and general information. The next step has been an integration of distributed factual knowledge with an ontology as its conceptual backbone. Thereby, we relied on the system Ontobroker. Indeed, Benjamins, Fensel, & Pérez (1998) already outlined how Ontobroker could be used for knowledge management. However, in their approach the user had to bear all the burden of doing the right things at the right time, while our approach goes in the direction of telling the user what might be useful for him in his very next task. A central point in our approach is reasoning about document structure and contents. Here we take over W. Weitz's view of SGML documents in a workflow process (Weitz, 1998), but extend his approach to include the knowledge management side and the weakly structured parts of processes.

Nearest to our integration of workflow and knowledge management aspects are works by Huber (1998), Reimer *et al.* (1999), Ackerman & Mandel (1999), and Mahling & King (1999). Huber (1998) builds on a Lotus Notes intranet environment that lets the user define a simple ontology and small workflows. However, his approach is less principled and does not lend itself easily for modeling and process planing goals. In particular, he cannot query facts, not to speak of implicit knowledge, but only documents. Reimer *et al.* (1999) supports the user with particular tasks. For this purpose, they use rather rigid process structures that are build from declarative business rules. We, in contrast, leave all the decision with the user and try to provide him with information that might facilitate his problem solving.

Ackerman & Mandel (1999) describe an approach that hierarchically structures tasks and abstracts from different types of data collections in order to support the users in their purpose of analyzing astronomical data. Thus, they pursue a goal that is comparable to ours. However, their application is much more dedicated to their particular goal. With our approach we intend to reach a higher degree of flexibility as far as the task goals are concerned.

Mahling & King (1999) describe an intelligent planning system that supports goal-based workflow similar as in our approach. For this purpose, they also devise an elaborate agent architecture such that electronic or human workflow participants may easily cooperate. Their approach however lacks an adequate level of description for the knowledge in the documents. Hence, the knowledge base of their system does not grow with its use, such as we require for the typical knowledge worker.

We share many of the convictions we build on with Leake *et al.* (1999). In their approach, they also aim at seamless interaction in task-based knowledge management. They provide an integration of various knowledge sources and some "*proactive*" support. We put *proactive* between quotation marks, since they use an approach that is more like our *Reactive Agent Support* where *queries*, rather than appropriate *answers*, are compiled proactively. Also in contrast to our approach, they use information retrieval and case-based reasoning techniques. This may appear advantageous, because their system may perhaps degrade somewhat more gracefully when its is given over- or underspecific queries when compared to our reactive agent support. However, the drawback they incur is that their scheme is not semantically based and, hence, may not provide similar semantic rigorosity, semantic derivations, and semantic-based compilation of appropriate answers.

Our approach builds heavily on considerations by Abecker *et al.* (1998) who establish a common grounding for documents, organization and knowledge (reflected by their information, enterprise and domain ontologies, respectively), but they do not go as far in drawing inferences and using this knowledge for a push technology and a tight integration of workflow and knowledge management, such as we do. We differ from common workflow management and office information support system by considering the document semantics in detail. In particular, we model document structures in order to provide better process support through inferencing on document knowledge. This support is not restricted to rigidly-structured processes, but it may easily be exploited for weakly-structured processes, too, where only parts of the overal order of task decomposition is known.

Conclusion

We have here presented an approach for intelligent, proactive inferencing agents that subsumes our earlier work on desk support that combined an organizational memory with business process modeling (cf. Staab & Schnurr (1999)). The reasons for the extension of the earlier approach stem from a comparison of the agent with a (very roughly) corresponding human assistant. We have made the experience that our first approach exhibited a lack of flexibility and proactivity in adding help to the project management setting or comparable tasks, while in the improved scheme presented here, the modeling of interesting queries is facilitated. This is due to the fact that careful balancing between overly specific and overly general questions may be delegated to the agent and need not be hand-crafted as before.

In this paper, we have again focused on the problem of project management. However, this does not mean that we are restricted to this scenario. As one may easily see, the — possibly digital — travel agent that supports you in your booking of your next vacation trip has to deal with the very same problems. It requires a complex memory with an ontological structure for all different types of housing and transport. An intelligent, supportive agent would proactively think about what you are doing. If you try to get a flight during Christmas season it might proactively determine that no matter what airline or exact date you choose, there are just no flights available for the cheapest airfare anymore. Current systems badly lack this type of proactive reasoning, as one of the others had to experience recently. The importance of such an approach may then be derived by the time a particular project manager saves in executing such a task, the more impressive and important consequences, however, will be derived from customer satisfaction as one of the key factors in enterprise services.

Acknowledgements

The research presented in this paper has been partially funded by the German Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie (BMBF) under grant number 01IN901C0 (project "GETESS; `http://www.getess.de`).

References

Abecker, A.; Bernardi, A.; Hinkelmann, K.; Kühn, O.; and Sintek, M. 1998. Toward a technology for organizational memories. *IEEE Intelligent Systems* 13(3):40–48.

Ackerman, M., and Mandel, E. 1999. Memory in the small: Combining collective memory and task support for a scientific community. *Journal of Organizational Computing and Electronic Commerce*. accepted for publication in 1999.

Benjamins, V. R.; Fensel, D.; and Pérez, A. G. 1998. Knowledge management through ontologies. In Reimer (1998), 5.1–5.12.

Boddy, M. S. 1991. Anytime problem solving using dynamic programming. In *AAAI-91 — Proceedings of the National Conference on Artificial Intelligence*, volume 2, 738–743.

Decker, S.; Erdmann, M.; Fensel, D.; and Studer, R. 1999. Ontobroker: Ontology based access to distributed and semi-structured information. In *Database Semantics, Semantic Issues in Multimedia Systems*. Boston, MA: Kluwer. 351–369.

Decker, S. 1998. On domain-specific declarative knowledge representation and database languages. In Borgida, A.; Chaudri, V.; and Staudt, M., eds., *KRDB-98 — Proceedings of the 5th Workshop Knowledge Representation meets DataBases, Seattle, WA, 31-May-1998*.

Erdmann, M., and Studer, R. 1999. Ontologies as conceptual models for XML documents. In *Proceedings of the 12th Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW'99), Banff, Canada, October 1999*.

Fensel, D.; Angele, J.; and Studer, R. 1998. The knowledge acquisition and representation language KARL. *IEEE Transcactions on Knowledge and Data Engineering* 10(4):527–550.

Huber, H. 1998. Document research based on collaborative provided structure knowledge. In Reimer (1998), 11.1–11.9.

Kifer, M.; Lausen, G.; and Wu, J. 1995. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM* 42.

Klusch, M., ed. 1998. *Intelligent Information Agents — Agent-Based Information Discovery and Management on the Internet*. Berlin, Heidelberg: Springer.

Leake, D.; Birnbaum, L.; Hammond, K.; Marlow, C.; and Yang, H. 1999. Task-based knowledge management. In *Exploring Synergies of Knowledge Management and Case-Based Reasoning. Proceedings of the AAAI-99 Workshop*, 35–39. AAAI.

Liebermann, H. 1998. Personal assistants for the web: An MIT perspective. In Klusch (1998). 279–292.

Lloyd, J. W., and Topor, R. W. 1984. Making Prolog more expressive. *Journal of Logic Programming* 1(3).

Mahling, D. E., and King, R. C. 1999. A goal-based workflow system for multiagent task coordination. *Journal of Organizational Computing and Electronic Commerce* 9(1):57–82.

Reimer, U.; Margelisch, A.; Novotny, B.; and Vetterli, T. 1999. Eule2: A knowledge-based system for supporting office work. In *CoopIS-99: Proceedings of the 4th International Conference on Cooperative Information Systems, Edinburgh, UK, September, 1999*.

Reimer, U., ed. 1998. *Proceedings of the 2nd International Conference on Practical Aspects of Knowledge Management, Basel, Switzerland, October 29-30, 1998*.

Staab, S., and Schnurr, H.-P. 1999. Knowledge and business processes: Approaching an integration. In Dieng, R.; Decker, S.; Matta, N.; and Reimer, U., eds., *Proceedings of the IJCAI-99 Workshop on Knowledge Management and Organizational Memory, Stockholm, July 31 - August 6, 1999*.

Weiss, G. 1999. *Multiagent Systems — a Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA: MIT Press.

Weitz, W. 1998. Combining structured documents with high-level petri-nets for workflow modeling in internet-based commerce. *Journal of Cooperative Information Systems* 7(4):275–296.